Leveraging hierarchical memories for micro-core architectures

Leveraging hierarchical memories for micro-core architectures

Nick Brown,[1] Maurice Jamieson[1]

[1] *EPCC, The University of Edinburgh, Scotland,* n.brown@epcc.ed.ac.uk, maurice.jamieson@ed.ac.uk

1. Background

Micro-core architectures combine many simple, low power and low on-chip memory cores onto a single processor package. The low power nature of these architectures means that there is potential for their use in future HPC and embedded systems, and their low cost makes them ideal for education and prototyping. However there is a high barrier to entry in programming due to the considerable complexity and immaturity of supporting tools.

ePython is a Python virtual machine we have developed for the 16-core Epiphany III micro-core architecture which fits in the 32Kb per core memory. In combination we developed an abstraction that supports offloading functions in existing Python codes, running on a host CPU, seamlessly to the micro-cores. In [1] we introduced this abstraction and motivated it with a machine learning code for detecting lung cancer in 3D CT scans where kernels for model training and inference ran in parallel on the micro-cores. However the small amount of core memory severely limited the physical size of the images, which had to be interpolated to fit.

2. Hierarchical memories

In addition to the small on-core memory, there is typically much larger, slower external memory as illustrated for the Epiphany by figure 1. In order to take full advantage of these architectures one must leverage these hierarchies of memory, but a key question is how best to achieve this whilst maintaining good performance.

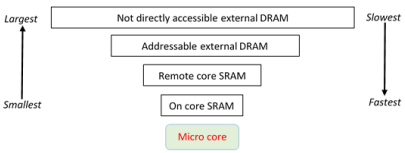

Figure 1. Typical Epiphany memory hierarchy

In this work we have addressed this challenge by splitting the memory abstraction into three choices:

- A mirroring of memory where copies of external memory also exist on the micro-cores. A manually copying of data to and from the different memory levels is required.
- Memory can be exposed from a specific level in the hierarchy to the micro-cores without an explicit copy being allocated. Reads and writes directly access this external memory, these accesses being blocking or non-blocking (using the DMA engines.) Abstractions around the non-blocking approach enables the programmer to leverage patterns such as double buffering and data streaming to overlap compute and memory access for performance.
- By default memory belongs to the hierarchical level where it is first declared. It is possible to override this via memory kinds [2]. In our approach these are Python objects that follow a standard interface and sit outside of the core ePython implementation. They define the behaviour of memory access at the level of hierarchy they represent.

Based upon this work we are now able to run the machine learning code of [1] with the full sized images. The programmer is able to experiment with choices around memory placement and access patterns without having to worry about the low level complexities of data movement.

References

[1] Brown, N. Offloading Python kernels to micro-core architectures, Poster Supercomputing 2017, Denver US

[2] Cantalupo, C. et al. Memkind: An Extensible Heap Memory Manager for Heterogeneous Memory Platforms and Mixed Memory Policies. No. SAND2015-1862C. SNL-NM, Albuquerque, NM (United States), 2015.